cRTOS: A Linux-compatible compounded RTOS based on NuttX, Linux and Jailhouse

Chung-Fan Yang
Fixstars Corporation
chungfan.yang@fixstars.com

August 15-16 2020
NuttX Online Workshop
About the Speaker

• Chung-Fan Yang
• Creator of the x86-64 port of NuttX
• From Taiwan, working in Japan
• Software Engineer at Fixstars Corporation
  • Software / Hardware based optimization, acceleration
• Hobby:
  • Embedded system
  • Poking around system software
Outline

• Introduction – What is cRTOS?
• Implementation
• Handling System calls
• Performance of cRTOS
• Demo
• Issues & Discussions
Academic publication

• Developed during my years in University of Tsukuba, Japan
• “Obtaining hard real-time performance and rich Linux features in a compounded real-time operating system by a partitioning hypervisor.”
  • Chung-Fan Yang and Yasushi Shinjo. 2020.
  • In Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments
  • DOI: https://doi.org/10.1145/3381052.3381323
Introduction

What's cRTOS?

NuttX Online Workshop
What is cRTOS(compounded RTOS)

1. Run a General-Purpose OS (GPOS) and a Real-Time OS (RTOS) together with a hypervisor
2. Snap them together as one big OS.
3. Run processes on this big OS.
   • Have access to both the benefits of the 2 OSs
   • Rich features of GPOS and Real-timeness of RTOS
4. User benefits from this easily programable real-time environment
System Overview

- **Normal realm – Linux**
  - Manages Non-RT devices
  - Soft real-time IRQ path

- **Real-time realm – NuttX**
  - Manages RT devices
  - Hard real-time IRQ path
  - Access Linux features with shared memory
2 different viewpoints on benefits

- Benefits of cRTOS against other real-time extensions for Linux
  - Hard real-time is possible
  - No patching Linux kernel → very maintainable
  - Program real-time with Linux API! → very easy to use

- Benefits of the Linux extension for NuttX (which is a part of cRTOS)
  - Let you execute (any) Linux programs in NuttX
  - No re-compiling, editing binary
  - Glibc and other libraries is usable
  - X window GUIs!
Concept of Rich real-time process

- Written with POSIX API and threads
- Executes in NuttX and Linux
- RT and non-RT threads
  - RT threads:
    - Contain RT algorithms
    - Interact with NuttX and RT devices
    - E.g. Timer, CAN bus, SPI, I2C
  - Non-RT threads:
    - Interact with Linux and Non-RT devices
    - Use rich features of GPOS
    - E.g. X window, TCP/IP
Shadow process

- Each rich RT process has a shadow process.
  - In the Linux as user process
    - 1:1 thread mapping.
    - Executes Linux system calls on behalf of Rich RT process.
  - Memory
    - Shared physical memory.
    - Same memory address space.
    - The same data at the same address in both process.
Shadow process

• Each rich RT process has a shadow process.
  • In the Linux as user process
    • 1:1 thread mapping.
    • Executes Linux system calls in behalf of Rich RT process.

• Memory
  • Shared physical memory.
  • Same memory address space.
  • The same data at the same address in both process.

Very important attribute, will come back later
NuttX Online Workshop

Implementation

POSIX is great

NuttX Online Workshop
Jailhouse - Creating 2 realms

• Current implementation used Jailhouse hypervisor[1]
  • Like Xen a Linux based Type-I hypervisor
    • Linux only as the bootloader and management interface
  • Partitioning hypervisor
  • Hardware resources are not shared.
    • No scheduling on vCPUs
    • Static memory allocation, might be shared
    • PCI-E device passthrough
  • Easily achieves hard real-time and feasible to runs RTOSs

[1] https://github.com/siemens/jailhouse
Linux – Normal realm with rich features

Well, it is the standard Linux everyone knows, nothing special.

No patching

Only 2 kernel modules for shared memory access
NuttX – Real-time realm

• Runs as another guest on Jailhouse
• Runs Linux program binaries
• Exploited the fact that
  • NuttX is POSIX confirming, so is Linux (mostly).
  • On source level, portable *nix program should work out of box.
  • System call set are very similar, main barrier is the ABI and VM (and the non-standard system calls which Linux had screwed up).

• Provide a Linux compatibility layer, Whoosh, Linux program binaries should work.
NuttX – Real-time realm

- Runs as another guest on Jailhouse
- Runs Linux program binaries
- Exploited the fact that
  - NuttX is POSIX confirming, so is Linux (mostly).
  - On source level, portable *nix program should work out of box.
  - System call set are very similar, main barrier is the ABI and VM (and the non-standard system calls which Linux had screwed up).

- Provide a Linux compatibility layer, Whoosh, Linux program binaries should work.

Development Goal
X86-64 NuttX

• By product of cRTOS, already merged to mainline
  • Try it and help report bugs!

• Jailhouse only support x86-64 and AArch64
  • And I happened to only have an x86-64 machine for development

• To make a Linux ABI compatible NuttX on x86-64
  • Ported NuttX to x86-64 with SystemV ABI
  • 50% done by compiler (Calling convention)
  • 50% hand coded (System call handler, XCP register set, FPU setting)
NuttX for Jailhouse

• Also a by product of cRTOS, already merged to mainline
  • Help testing!
• It can be used separately.
• Shared memory driver is implemented
  • Not yet merged.
  • PCI driver framework need to go first.
  • GPL license issue, need full rewrite.
System Overview

Compounded RTOS

Normal realm
Linux

Real-time realm
NuttX

Share Memory

Partitioning Hypervisor

Non-RT Core 1 2 ... N
PIC
Non-RT Device

RT Core
PIC
RT Device

PIC: programmable interrupt controller
Handling System calls

Linux compatibility Layer

NuttX Online Workshop
Extending NuttX for Linux style process

- NuttX has some degree of protected or kernel build.
  - But quite far from a Linux compatible environment

- For simplicity,
  flat build is chosen and extended to support Linux style process.
  - Virtual memory supported is implemented.
  - Like Linux, kernel is in mapped to high memory
  - Process occupies lower memory
  - Dynamic memory mapping supported is added (mmap / munmap)
  - No actual protection between kernel and user space memory
Extending NuttX for Linux system calls

• Impractical to implement every Linux system call in NuttX
  • The existing system calls in NuttX cover a good variety of real-time usage

• We need a way to get over those
  • Nasty Linux specified system calls
  • System calls inessential to real-time

• We try to delegate those not important system calls to side-by-side Linux
  • Gives an excellent coverage
  • Trade-off between hard and soft real-time
System call handler

• Reuse the system call reservation mechanism
  • Lower 512 system calls are reserved for Linux system calls
  • Effectively moved NuttX system calls to 512~

• For 512~ calls, continues to function as-is
  • Native NuttX apps continues to function properly

• For 0~512 calls, either
  • In Nuttx → Real-time system call
  • Delegating to side-by -side Linux → Remote system call

• The selection of delegating or not is seamless, user code uses standard system call exception interface.
Real-time system calls

• Real-time related system call will be handled locally in NuttX.
  • Deterministic execution
  • Higher timing stability

• Access to local facilities
  • Synchronization: semaphore, shared memory, etc.
  • Etc.

• Access to RT devices
  • open, read, write, etc.
Remote system calls (RCSs)

- RCSs provide access to Linux features seamlessly
  - Access to non-RT devices, file systems, credentials
- Delegated system calls to Linux as messages via a queue.
  - Executed by corresponding shadow process
  - For handling pointers, shadow process shared same memory space
Overlay FS

- Open system call try NuttX files first before trying Linux files
- Effectively produce an Overlay FS

- The returning file descriptors is segregated, allow multiplexing
  - 0~4096: Linux files
  - 4096~: NuttX files
Extending NuttX system calls

• Nonetheless, some of the system calls
  • Doesn’t exist in NuttX
  • Cannot be simply delegated to Linux because of semantics problem

• For example:
  • Process / threading related: clone, fork, arch_prctl, etc.
  • Memory management: mmap, munmap, etc.
  • SystemV IPC: shm, etc.
  • Timer: alarm, timer_create, etc.

• Implemented those system calls
  (A lot less comparing to all of Linux system calls)
  • Most of them are stubs and wrappers
Dual system calls

• Among the extended system calls, some are dual system calls
  • Executed in both NuttX and Linux

• Synchronize the attributes between rich real-time and shadow process.
  • Memory map
  • 1:1 thread relationship

• Clone, fork, exit, mmap, munmap, exec are implemented as dual system calls
Starting a rich real-time process

• A daemon executes on Nuttx

• A loader program
  • On Linux side
  • Makes a remote exec call to the daemon on NuttX side

• The daemon creates a seed rich real-time process
  • The rich process calls exec system call to start the user appointed program.
Performance
First direct comparison of NuttX and Linux ever?

NuttX Online Workshop
# Environment

## Hardware

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon 2650 v4 @ 2.2Ghz 10C/10T</td>
</tr>
<tr>
<td>RAM</td>
<td>32GB DDR4</td>
</tr>
</tbody>
</table>

## Software

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jailhouse version</td>
<td>v0.9.1</td>
</tr>
<tr>
<td>Linux kernel version</td>
<td>v4.9.84</td>
</tr>
<tr>
<td>Nuttx version</td>
<td>v7.2</td>
</tr>
</tbody>
</table>

## Configurations

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla Linux</td>
<td>PREEMPT_RT</td>
</tr>
<tr>
<td>Proposed cRTOS / w vanilla Linux</td>
<td>Proposed cRTOS / w PREEMPT_RT Linux</td>
</tr>
<tr>
<td>Xenomai 3.0</td>
<td></td>
</tr>
</tbody>
</table>
Cyclictest

- Cyclictest:
  - A thread set a timer and the timer expires.
  - Measures the elapsed time for accuracy.

- All configurations used the same binary,
  - Xenomai required a modified version of cyclictest.

- Parameter for cyclictest:
  - SCHED_FIFO, priority 90, interval 1ms, loop 100k times

- STREAM benchmark suite was used as extra load for hardware.
Cyclicttest

- The performance of real-time realm(NuttX) was the best
  - Latency: 4 us max / 4 us jitter
- Performance became better with PREEMPT_RT

![Graph showing latency and performance comparison]

- Smaller is better
- 100000 measurements
I/O Interrupt latency

- We measured the latency of a hardware interrupt.
- A serial device was attached to each configuration.
- The system was programmed to generate an output upon an input is received.
- The gap between 2 pulses were measured with an oscilloscope.
I/O Interrupt latency

• The performance of cRTOS beats all other solution
  • Latency: 10 us max / 2 us jitter
• cRTOS’s performance became better with PREEMPT_RT

![Diagram showing latency comparison between different operating systems and configurations](attachment:image.png)

- Larger is better
- Smaller is better

1000 measurements
System call latency

• Tested with original syscall micro-benchmark from Lmbench.
• Real-time system calls are faster than native Linux system calls.
  • vs PREEMPT_RT: over 4 times faster
• Remote system calls are quite slow

Table 1. The maximum latency of various system calls. Measured by Lmbench in microseconds.

<table>
<thead>
<tr>
<th>Environment</th>
<th>getpid</th>
<th>read</th>
<th>write</th>
<th>open and close</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREEMPT_RT native</td>
<td>0.306</td>
<td>0.406</td>
<td>0.338</td>
<td>2.23</td>
</tr>
<tr>
<td>Xenomai 3</td>
<td>0.456</td>
<td>1.14</td>
<td>1.07</td>
<td>4.16</td>
</tr>
<tr>
<td>Real-time system call</td>
<td>0.059</td>
<td>0.088</td>
<td>0.083</td>
<td>0.445</td>
</tr>
<tr>
<td>Remote system call</td>
<td>—</td>
<td>27.7</td>
<td>27.0</td>
<td>56.3</td>
</tr>
</tbody>
</table>
X window Applications in Nuttx!

vim
Ristretto
Ghost script

Xterm /w dash
Image magick

Emacs
Gnome terminal
Gedit
Firefox

Ixardoscope
NuttX Online Workshop

Demo
Issues & Discussions
NuttX Online Workshop
License Issues

- GPL2 and BSD licensed code exist in current source tree.
- Jailhouse’s share memory driver
  - Ported from Linux (which is GPL2, of course)
  - Rewrite is required, but how much is enough?
- Linux system call interface headers, a.k.a. UAPI headers
  - Contains system call related C struct, enum, MARCO definitions.
  - Required to parse and translate flags and structure into NuttX form.
  - GPL2 with “user program” exemptions, but we are not a “user program” in Linux!
  - Will a rewrite will save us?
Future work

• Contributions are welcome
  • Require more people to test this on more boards and applications
  • Porting to AArch64?
    (Jailhouse and Linux is available, so it is very possible)

• Current maintained out of mainline

• Might make its way into the mainline
  • Prove such model is practical in use and beneficial for NuttX community
  • If the license issues are settled
Source Code:

- Hosted on the Github page of Fixstars
  - [https://github.com/fixstars/cRTOS](https://github.com/fixstars/cRTOS)

- Ported to Linux 5.4, Nuttx 9.1, Jailhouse 0.12
  - Open tickets if you find any issues!
Thank you!

Questions?

chungfan.yang@fixstars.com

Or the nuttx.event forum

NuttX Online Workshop